Microservice Architecture: Optimizing for speed

An in-depth article upon what Microservice Architecture stands for and with it what pros and cons it brings along.

**“Microservice Architecture is a term used to define the procedure of dividing up an application into a series of smaller and more specified parts, where each part communicates with another through common interfaces.”**

To know more about microservices we must know a bit of the background story.

# Background

In the early stages of computer revolution, using almost any computer required writing a custom software. Only a Ph-D in science and computer could use these programing languages and entry into these programing languages was a tremendous task.

In the 1960s, the use of computer application skyrocketed and thus in 1964, **Basic** (a general-purpose programming language) was developed lowering the barriers for the entry in the programming allowing students without Ph-D to write executable programs.

The growth also brought forth a complexity of Software System which was overcome by the old times method of **Divide and Conquer**

1970s brought Modular Software development through the works of people such as **Edsger W. Dijikstra** (who in 1972 introduced the concept of Separation of Concern) and David **Parnas** (for his idea of modularity and information hiding in software’s for his paper of 1972).

This gave the idea of decomposing a large, complex software system into **“Loosely coupled, highly cohesive”** modules which communicated via internal interfaces.

Loosely coupled mean the dependency between modules should be very low.

Highly cohesive means that the that mono module should focus on single or similar functionality.

The rise of the internet and web in the 1990s software became widespread in business applications and became even more complex and large. Although modularity is used to reduce the complexities of the software applications, but often it did not help as the soft modular boundaries of software sub-system are easy to cross and misuse. **Layered Architecture** was another software architecture pattern that became very popular in the 1990s to develop business applications.

An ideal business Web Applications is divided into several layers as shown below:

A screenshot of a cell phone

Description automatically generated Layered Architecture

As a result of Mobile internet and faster networks the late 2000s saw Cambrian Explosion. The software took over the world with a storm and all types of business started going digital such as Banking, Hotels, Music etc., and companies such as Facebook, Twitter, Uber, Netflix, Spotify came with such innovative, aggressive approach that the Monolithic Architecture could not handle the challenges being thrown at them. It was tine for a more efficient approach.

To know what need to improve we must know what the limitation were being faced.

# Limitations of Monolithic Architecture

## Application Scaling

As the successful Web Scale companies see exponential growth their software isn’t able to keep up with their need for support high horizontal scalability and where monolithic software works as a single unit and developed in a single programming language using a single Tech Stack an architecture was needed that could support the polyglot programming .

A close up of a device

Description automatically generated

If horizontal scaling is desired then the whole application needs to be scaled and with monolithic software only supporting one programming language, we are not able to implement one single module of it in other programming languages.

## Development Velocity

In the new fast paced world, the companies are for ever more wanting a shorten time to market thus wanting a fast feature development which unfortunately for an Monolithic Application is very slow because such applications gives huge cognitive load to the developer as the modules of giant monolithic applications are tightly coupled adding another challenge to the developer thus adding cost to the development. All in all it becomes very time and cost expensive to add new features

## Development Scaling

Now a day’s companies have several new projects in the development parallel thus called parallelizing development. This being s highly desired, companies higher more and more developers for fast pace development. However, the problem arises when a developer meets a monolithic as he/she cannot work on it independently due to the fact it needs extra synchronization being a tightly coupled code. Thus, adding more developer doesn’t help produce new feature and becomes a liability on the company. Similarly, due to cognitive load, new hires / fresh graduates take longer to write the first piece of productive code.

## Release Cycle

For a monolithic application a release cycle is usually form any where between 6 months to 3 years. The technical advancement today brings forth new development every few months. For competitive companies a large release cycle can put the company under tremendous disadvantage for new companies can take over the market during the development period.

## Modularization

In monolithic Architecture, the modules communicate between internal interfaces. Development causes the application to grow and thus the communication aka the boundary between the modules falls apart. As a result, often modules in monolithic architecture become tightly coupled instead of being “loosely coupled, highly cohesive.”

## Modernization

Existing successful applications needed to be modernized due to many factors (e.g. taking advantage of modern Hardware, Browser, Network Bandwidth, Tech Stack or Attract good developers). Modernization of Monolithic application is often expensive and time-consuming as it needs a Big Bang modernization of the whole application without disrupting the Service.

# Microservice Application

Finally, we move on the topic in spotlight. Lets first look at the definition:

“Microservice *Architecture is about decomposing a Software System into autonomous Units which are independently deployable, and which communicates via lightweight, language agnostic way and together they fulfill the business goal.*”

In the 2010 the arise of such disruptive technology came into being which impacted the Software development landscape in a significant way (these included cloud computing, Containerizations, DevOps). Similarly, some new highly productive lightweight programming languages came to light including Rust and Swift. Some were easier to use than others such as JavaScript, Python became mainstream.

Waterfall software development module was almost discarded replaced almost immediately by a fast, iterative, incremental software development methodology:

**Agile Software development.**

Computer hardware upgraded to a new faster more cost-effective pieces with the rise of Multi-core CPU and GPU. The whole world was changing digitally with Database technology like NoSQL and NewSQL emerging and becoming more mainstream

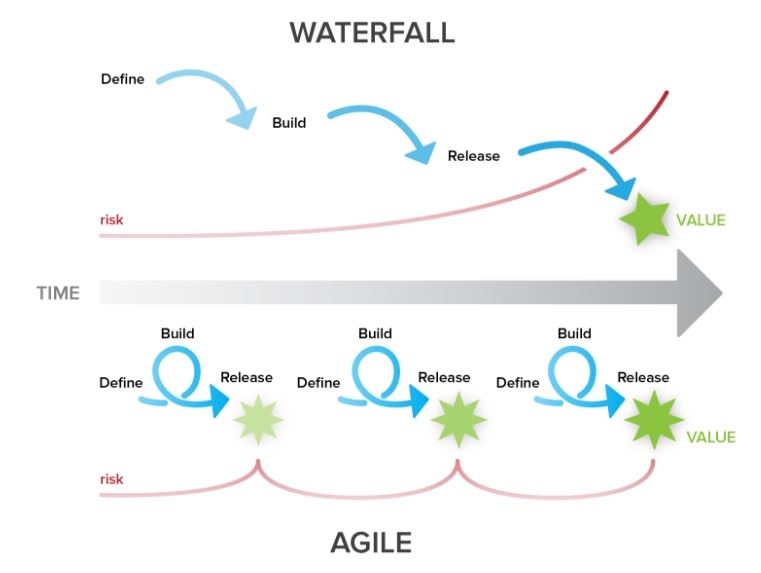
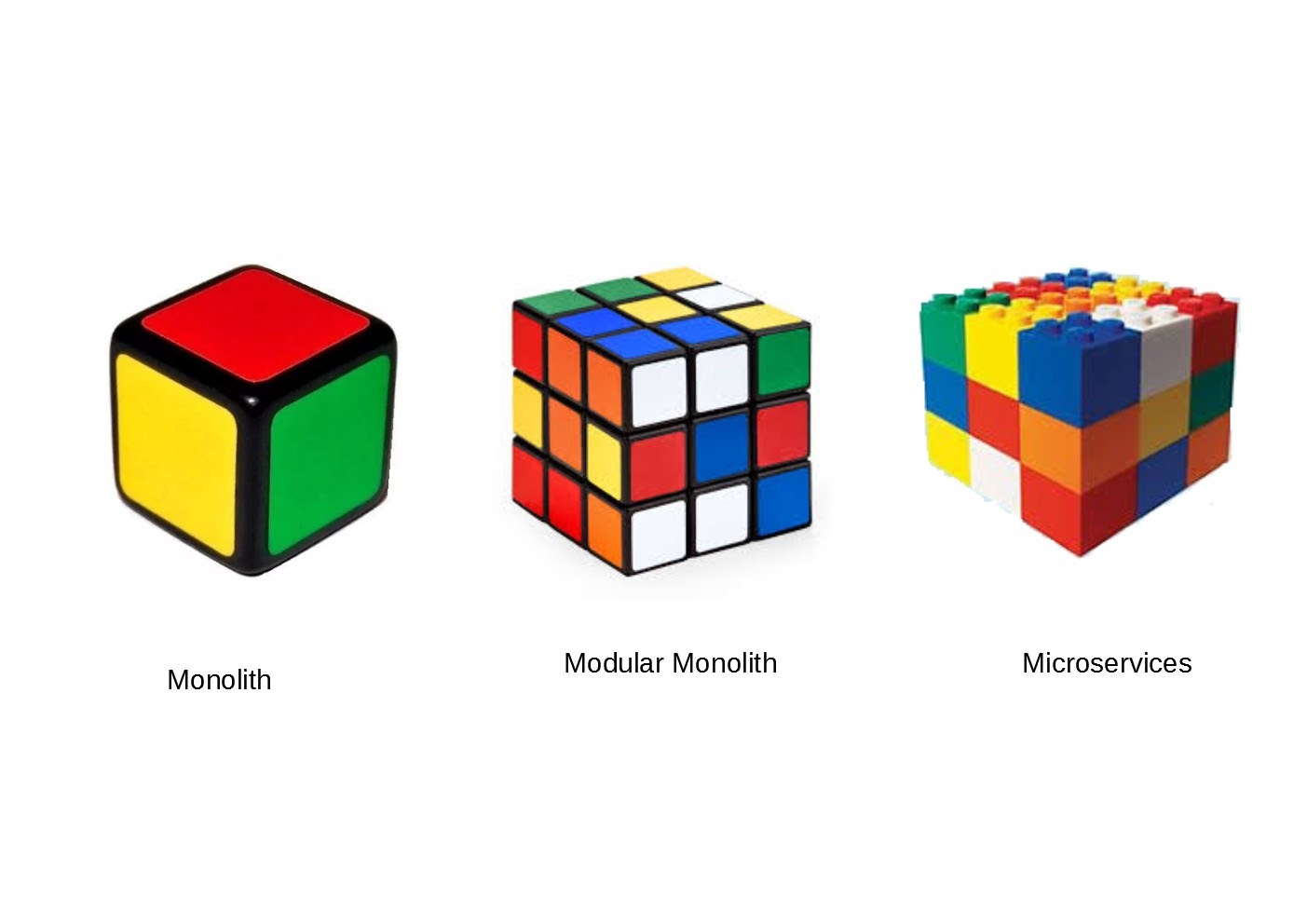


Figure 1- Difference between waterfall and agile methodology

In 2012, a new software architecture (to fulfill the need of modern software development : fast development, horizontal scaling ) took over called : Microservices Architecture which came over to handle the complexity of modern software applications, is able to take the advantage of Cloud Computing, Containerization and DevOps. It even allows the company to benefit with modern programming languages.

But how you may ask, is it able to do all these tasks? It uses the same technique of divide and conquer likewise with the monolithic architecture, what differentiate these two is that the microservices can be deployed independently whereas all the modules of monolith must be deployed as whole.



Think of it like this. The image above shows the various architecture. Monolithic application is one single unit (tightly coupled) like a single Cube. Modular application is like Rubric Cube which can contain small modules, but the modules cannot be separated, is interchangeable and can only be deployed together. Microservices are like a Lego cube made by Lego blocks where the blocks are loosely coupled, easily separable and changeable, thus all the Lego blocks together made the cube.

Lets look at what makes this architecture so desirable to the tech Giants.

# ADVANTAGES OF µSERVICES

## Application Scaling

The main attraction is the high horizontal scaling, bringing tech Giants to µservices. If executed carefully, µservice provide horizontal scaling within seconds. What’s more it even supports ployglotting, for say if a microservice is CPU intensive it can be implemented in CPU optimized programming language and likewise other µservices can be implemented in different programming languages.

## Development Velocity

When developing, a company wants the shortest window between developing and market. µservice being a quite small, adding new features are usually faster thus making the whole development process more time efficient and more desired by companies.

## Development Scaling

Microservices are autonomous and can be developed independently thus making it easier for the developing team to work at different aspects of the applications simultaneously and autonomously. Hiring more developers becomes more efficient to the company as it helps scale the development. The main thing being the cognitive load on new hires is low which in returns help the developers write first piece of productive code for the company.

## Release Cycle

Being independently deployable the release cycle is shortened significantly such that with the use of CI/CD pipelines, it is possible to give several releases per day compare that to the monolithic architecture it is a huge step forward.

## Modularization

The problem being faced with was that the boundary between the modules were dissolved as the applications grew, µservices has the advantage of external interface (aka physical Network) instead of internal which is hard to cross. Thus, µservice offers the ideal modularization “Loosely *coupled, highly cohesive”.*

## Modernization

Microservice are usually loosely coupled and communicate via language-agnostic way, hence if the need arises these µservices can be replaced easily by the developers using a new programming language. It brings forth the ideal modernization which is always incremental and not big bang

Like any other architecture, program and software microservices are not perfect. The following are the disadvantages of microservices architecture.

# DISADVANTAGES OF µSRVICES

As like anything in life, microservice architecture has also its price and a fair share of disadvantages. It is by no means a Golden Hammer which can solve all sort of Problems in a Software Application. There are scenarios in which moving to µservice architecture from monolithic architecture without proper consideration will leads to nightmarish condition

## Design Complexity

An advantage for monolithic architecture was that it gave the “one size fits all” solutions for Business applications. But this is not seen in µservices architecture as it provides variable solutions to applications and its use cases. If the solution is not suitable for the given scenario the architecture fails.

The complexity also carries on in the designing as there more interchangeable parts compared to monolithic architecture, this becomes dangerous as a carelessly developed µservice design is worse than a monolith.

## Distributed System Complexity

## Microservices are distributed system, which are complex and has a unique set of challenges compared to single Machine systems

## Following problems can arise in Distributed Microservices:

## Overall System latency is higher

## Network failure or Individual Node failure can bring the whole system down

## Operational complexities are higher

## Security

## Security is one of the major issues of µservices as securing one software application is a major task on its own, however securing several µservices which are often distributed system becomes challenging task.

## 

## Operational Complexity

## Once the application transforms from monolith to microservice, simplicity of the whole application is overtaken by complexity as it moves from the source code to the basic operations of the applications. Logging, monitoring and such simple operation take a turn for worse and become more difficult to manage as the number of systems needed to be handle increases compared to just one in monolith.

## Along side such complexity we need to keep tracing to measure the latency/performance of individual microservice for a service request. Hence in the final complete system test is equally more complex as each module needs to be tested compared to the monolithic application

## 

## Data Sharing and Data Consistency

Ideally every µservices should have its own data storage. This does not become the case when the µservices need to share data between themselves to achieve a business goal.

Data consistency is another challenge as to support the consistency in the distributed database is not recommended for the following reasons:

1. It does not Scale and many Modern Data Store does not support it
2. Most of the modern NoSQL database only offers Eventaul Consistency which needs careful design.

## Communications Complexities

µservices achieves strict modularity and development autonomy via process/ network boundaries. Downside being the µservices can only communicate via the physical network which eventually leads to higher network latency

# CONCULSION